LEMAN: A System for Constructing and Animating Layered Elastic Characters

Russell Turner
University of Maryland, Baltimore County

ABSTRACT

An interactive animation system is presented for constructing layered character models with simulated elastic components. The system, called LEMAN (Layered Elastic Model ANimation), allows three-dimensional animated characters to be built up from successive layers of skeleton, muscle, fat and skin in a completely interactive, direct-manipulation environment, using a variety of input devices. Using an artist's anatomical approach, the character is represented as a simulated elastically deformable skin surface which is wrapped around a kinematically modeled articulated figure. It may then be animated by moving the underlying articulated figure, either interactively using forward or inverse kinematics, or by interpolating a sequence of key postures. Once a motion sequence has been specified, the entire simulation can be recalculated at a higher surface resolution for better visual results. Although the system is most practical for animating relatively simple cartoon-like characters, the realistic-looking shapes and movements resulting from the physical simulation make it well-suited for research into naturalistic human and animal animation.

Keywords: Character Animation, Physically-Based Models, Deformation, 3D Interaction.

1. INTRODUCTION

Practical three-dimensional character animation requires interactive tools for both construction and animation of characters. Whether animating simple caricatures or realistic-looking humans, character animation is an essentially creative process and it is necessary that the software tools be accessible to non-technical, creative animators. Ideally, the computer should provide for the traditional sculptor or animator an artistic medium as flexible and accessible as any traditional medium such as clay or pencil. Recent advances in 3D graphics display and input hardware bring the possibility of building a virtual animation environment for this purpose, in which the user can become immersed in and directly interact with the character as it is being sculpted or animated.

Layered construction techniques which model anatomical features have shown promise in creating character models that deform automatically around an articulated skeleton. But purely geometric models, although they can be very expressive, usually require too much user intervention to achieve realistic-looking results. Physically-based elastic models provide more realistic behavior at the price of CPU resources and difficulty of control. With proper use of constraints, however, deformable models can be controlled by kinematic geometrical models. Recent improvements in processor speeds now make it possible to simulate certain kinds of moderately complex physical models in real-time.

For these reasons, we believe that a hybrid approach in which layered models are constructed using a combination of geometric, kinematic and physically-based techniques, is the most promising one. The ideal 3D character model should provide a good compromise between interactive speed and realism, and between animator control and physically realistic behavior. The exact details of such a model are no more important, however, than the types of interactive technique used to construct and animate it. High-performance 3D graphics workstations and variety of multi-dimensional input devices have begun to make highly interactive, direct manipulation environments practical. Finding the right kinds of interaction metaphors by which these devices can control a 3D character model, however, requires experimentation with many of the various possibilities. This paper will describe the LEMAN system, originally developed at the Computer Graphics Lab of the Swiss Federal Institute of Technology, which can be used to construct and animate 3D characters based on the elastic surface layer model in such an interactive, direct-manipulation environment.

2. PREVIOUS WORK

The simplest kinds of layered character models contain a skeleton and outer skin layer. Points on the skin surface are bound to individual skeleton joints, and various geometrically-based techniques can be used to handle deformation at the joints [Magnenat-Thalmann 1988]. This division into two layers is conceptually simple and is also practical, since the skin envelope can be sculpted using standard surface modeling techniques, or scanned directly from a sculpture or a living creature. Simple observation of human or animal skin in motion, however, reveals that the deformation of the outer skin envelope results from many other factors besides the position of the skeleton. In fact, the skin reveals as much as it conceals details of the underlying anatomy in motion. More advanced layered models take into account some aspects of these intermediate anatomical layers between the skeleton and skin.

Some 3D character models, such as the dinosaurs in *Jurassic Park*, have used geometric techniques to simulate underlying muscle and bone layers. To improve realism, and to create more interesting deformations, layered elastic models are used which add physically based elastic components to some or all of the layers. A simple example of this type of approach is Pacific Data Images' *Goop* system, in which a mass and spring with damping are attached to each vertex of a polygonal model [Walters 89]. Moving the model is accomplished by moving the anchor points of the springs, causing the vertex points to oscillate about the new position until the motion is damped. Gavin Miller took this idea a step further by attaching the vertex points together to form a mass-spring lattice. This was arranged in the form of a tube to model dynamic snakes and worms [Miller 88]. By actively varying the rest-lengths of the springs, he was able to simulate muscles, resulting in worm-like motion. This model did not have any internal skeleton, however.

A more sophisticated examples of layered elastic construction for animated characters is found in the Critter system [Chadwick 89] in which a regular network of connected springs and masses is used to create a control point lattice for free-form deformations of the geometric surface. Some of the control points are bound to links of the underlying skeleton so that, when the skeleton is animated, the unattached mass points are influenced to move through the spring lattice. Since the lattice is controlling the space deformation and not the actual model, it can be relatively coarse with few enough mass points that the physical simulation can be conducted at interactive speeds. This can be quite effective in producing large-scale deformations of the character, but the resolution of the deformation is limited by the size of the mesh, and although the mass-spring lattice allows for shape control over the body deformation, the skin is still fundamentally a geometric surface model, not a model of a physical skin.

An elastic layered model for complex deformable bodies was proposed by Gascuel et al [Gascuel 91] in which the control points for an interpolating spline surface are bound to a rigid bone layer by stiff springs. The control points are connected to each other to form a surface using a non-physically-based geometric technique for propagating deformations. Another type of elastic layered model in two dimensions was developed by Overveld [Overveld 90]. The finite element method was used by Gourret et al [Gourret 89], who describe a human hand modeled as a volume element mesh surrounding bones. This was used to simulate a hand grasping a rubber ball. Chen et al

[Chen 92], also used the finite element method to develop a biomechanically-based model of muscles on bone, without skin, based on biomechanical data from a frog's leg.

Terzopoulos, who has used elastically deformable models to simulate simple animals such as fish [Terzopoulos 94], used a layered elastic model to implement facial animation [Terzopoulos 91]. In this model, an elastic solid simulation, consisting of a mass-spring lattice of depth three, is attached to a human skull model and deformed by muscles which take the form of force constraints between points on the skin surface and the underlying bone. Volume preserving constraints are used to simulate the effects of incompressible fatty tissue. This model, which was implemented in a system running in near real-time, represents a promising approach to layered construction. The idea of wrapping an elastic surface simulation around a human form by using force constraints was originally developed to animate clothing [Carignan 92], and together with the Terzopoulos facial model forms the conceptual basis for the similar wrapping of a skin surface around a muscle layer in the elastic surface layer model.

3. IMPLEMENTING THE ELASTIC SURFACE LAYER MODEL

As described in detail in [Turner 93] the development of the elastic surface layer model is an attempt to simulate layered anatomical structure so as to minimize computational effort by using for each layer the modeling techniques which are most appropriate. Since the skin is the outermost layer and the only one directly visible, we concentrate CPU effort on this by modeling it as a simulated deformable elastic surface [Terzopoulos 87]. The underlying layers are then modeled using geometric and kinematic techniques which act on the surface as force-based constraints. In particular, reaction constraints prevent the surface from penetrating the underlying layers, pushing the skin out, while point-to-point spring constraints pull the surface in.

The skin surface is implemented as a simulation of a continuous elastic surface discretized using a finite difference technique [Terzopoulos 87]. The surface is represented as a rectangular mesh of 3D mass points, together with their physical characteristics (e.g. mass, elasticity) and their current state information (e.g. position, velocity). When the numerical solver is turned on, the state is evolved over time at a fixed simulation time step. At periodic intervals, usually at least five per second, the surface is rendered on the screen, resulting in a continuous simulation at some fraction of real-time.

The surface is bounded at its poles and constrained by reaction constraints, point-to-point spring forces and other environmental forces such as gravity and air pressure. At each time step, the spring forces acting at each surface point are calculated, according to the hookian spring constant, and added to the total applied force for that point. Other environmental forces are then calculated and added in to the total applied force. Then the point is checked to see if it is inside any of the muscle layer surfaces, in which case reaction constraints are applied. Rather than simply adding forces, reaction constraints remove undesirable forces and replace them with forces that move the elastic surface towards the constraint surface with critically damped motion [Platt 88]. Forces perpendicular to the constraint surface are not affected, so the elastic surface may slide along the constraint surface until it reaches an energy minimum.

4. USER ENVIRONMENT

Unlike many research animation systems, the LEMAN system was intended to demonstrate a system usable by non-programmers, in particular, by professional animators. To them, the process of creating an animation should be as tangible and direct as modeling a piece of clay. Therefore, we have tried as much as possible to make the user interface follow an intuitive, direct-manipulation style, for example like the Macintosh, without any scripts, configuration files or other components that hint of programming. There is, therefore, a very firm boundary between what the user sees and what the programmer sees. This section will describe the ideal LEMAN system from the user point of view.

The animator sits in front of a high-performance Silicon Graphics Iris workstation which can display lighted, texture-mapped surfaces containing many thousands of polygons at interactive update rates. He looks at a window containing a single, lighted, perspective view into the three-dimensional world in which the animation is to take place. Surrounding this window, or perhaps on a separate monitor sitting next to him, is an arrangement of windows containing two-dimensional widget panels which can be used to issue command and adjust parameters of the animation. In one hand he holds a mouse, in the other a spaceball. Optionally, he may also have other types of multi-dimensional input devices next to him such as pressure-sensitive digitizing tables, a MIDI keyboard, as well as an audio system which can play synthesized or prerecorded digital sounds.

Starting from scratch, the animator can build up a three-dimensional character model, adjusting parameters and moving it interactively throughout the process to test its appearance and behavior. Once the character is constructed, he can save it to a file for future use. The character can then be animated using an interactive key frame animation technique and these animation can be saved as well. Any number of animation sequences can be created for a single character.

The LEMAN system allows layered elastic characters to be constructed and animated in a totally interactive, direct manipulation environment, using, multi-dimensional input devices such as the spaceball, valuators, and MIDI keyboard. Traditional desktop-metaphor interaction techniques are also available such as the mouse and widget panels. Such a variety of input techniques allows several possible working configurations. At one extreme is the familiar mouse-and-widget metaphor, usually using a virtual trackball for 3D interaction. From here, one can add other devices, such as the spaceball, valuators, MIDI keyboard, and dataglove.

Since most 3D input operations can be expressed in the form of either an absolute or a relative 4x4 homogeneous transform matrix, the various 3D input devices can more or less be interchanged at will. We therefore concentrate our software development efforts on creating a collection of 3D interaction metaphors, or *manipulators*, and let the user assign which 3D input device controls them at run-time. As a minimal configuration, we usually have two 3D input devices available: the spaceball and the virtual trackball, which is normally bound to the mouse. Most common operations can be performed using the spaceball with one hand and the trackball with the other. For example, the spaceball can be used to position and orient the character while the trackball can be used to move the joints. Less frequently performed operations are accomplished using the widget panels. If we wish to avoid using the widgets altogether, to make more screen real-estate available for viewing the scene, their functionality can be easily bound to keys on the MIDI keyboard or to dataglove gestures.

5. CONSTRUCTING A CHARACTER

Figure 1 shows a sequence of stages in the interactive construction process for an elastic surface layer character, in this case a Venus figure. Starting from scratch, the animator first builds up an articulated skeleton, then adds the muscle layer as links attached to the skeleton joints. Then an elastic surface skin is wrapped around the articulated figure and the physical simulation is started. Finally, the fat and connective tissue layers, which control the attachment of the surface skin to the underlying layers, is adjusted. The process is iterative, that is, the animator may step back to any point in the process without losing work. Figure 2 shows a final rendering of the character using a commercial animation system.

5.1. Skeleton Building

The skeleton is first built using the hierarchy-building tools, which provide basic operations on joints such as creation, deletion, copy and paste. A joint represents a 4x4 homogeneous space transformation together with a single rotational or translational degree of freedom. Articulated structures can be constructed by arranging the joints in hierarchies. The current joint can be selected using the mouse. Then, by using one kind of 3D interaction metaphor, the *joint manipulator*, the local transformation of the current joint can be moved to its desired position and orientation. With a 3D local coordinate snap-to-grid mode, this can be done precisely. Joints are represented as a red-green-blue coordinate axis with a hierarchy line drawn from its origin to the origin of its parent. A pure hierarchy of joints therefore resembles a stick figure skeleton.

A second interaction metaphor, the *inverse kinematic manipulator,* can be used to move the current joint as the end-effector of a kinematic chain. In this way, the skeleton kinematics as well as its structure may be tested at any point in the construction process. The inverse kinematic manipulator takes relative transformations of the end-effector (both translational and rotational) and multiplies them by the pseudo-inverse of the chain's Jacobian matrix to determine the differential joint angle values [Klein 83] When this manipulator is bound to the spaceball, for example, the chain can be directly manipulated in six degrees of freedom, as though the animator were simultaneously moving and twisting the end-effector.

The kinematic chain can be specified by setting a root joint, which is considered the base of the chain and then treating the currently selected joint as the end-effector. As an alternative, an automatic root selection mode can be set, which walks up the hierarchy from the current joint until the first branch point is found, setting this to be the root joint. Developing good interactive skeleton manipulation techniques is important because, once the character has been constructed, this constitutes the bulk of the work done by the animator to create the final animation. Since the motion can usually be visualized without displaying the skin surface, like a traditional pencil-test, this kind of interactive work can usually be done at very high screen update rates.

5.2. Adding Muscles

This stick-figure skeleton can then be fleshed out by adding muscle surfaces as links attached to the skeleton joints. Muscles are modeled as deformable implicit surfaces which prevent penetration by the skin layer. We currently use spheres, cylinders and superellipses [Barr 81] together with global deformation functions [Barr 84], but any implicit surface for which a rapid inside/outside function exists could be used. Since the muscle surfaces push the outer layers out via reaction constraints, it is important to be able to test rapidly whether a point is inside a muscle surface.

Muscles can be created by the animator by attaching a shape object to the currently selected joint. It is also possible to create complex hierarchical muscles by attaching link subhierarchies to the joints. These link subhierarchies are made from static nodes each of which can contain a shape object. Shape objects can be edited to control the type of shape (sphere, cylinder, superellipse), the shape parameters, and the global deformation parameters, which can be adjusted using sliders. Shapes can be made visible or invisible and active or inactive. Active shapes push the skin to be on the outside, while inactive shapes do not affect the skin surface at all. Visible, inactive shapes can be used to represent external, non-deformable components of the character.

In practice, building up the skeleton and muscle layers is usually done together in an iterative process. Figure 3 shows some stages of the skeleton and muscle building process for a human torso.

5.3. Attaching the Skin Surface

When the muscle surfaces have been added, the skin surface mesh (initially in a spherical shape) can be created and connected at each pole directly to points on the muscle layer. These attachments are fixed, geometrical constraints which anchor the skin to the skeleton so that it does not slide off. The polar points can be connected either automatically, or by the animator on a point by point basis using the mouse. It is also possible, once the simulation is running, to set the polar regions to any particular cross-section of the skin surface using a process of "walking" the polar attachment up or down the surface.

At this point, the numerical solver can be started and the initially spherical surface "comes to life" as it changes shape under the influence of the elastic simulation. Initially, the only force applied to the surface is a uniform internal air pressure force, which pushes the elastic surface outward like a balloon. The amount of pressure can be adjusted, increasing or decreasing the size of the balloon to ensure that the surface is completely outside the underlying surface layers.

5.4. Activating Reaction Constraints

Next, the animator can turn on the reaction constraints and slowly reduce the pressure until the surface shrinks around the articulated figure, much like plastic shrink-wrap. The reaction constraints push the skin surface outside the muscle layer, but leave it free to slide along it until the entire surface finds an energy minimum. The global nature of this behavior means that local changes to the surface properties have global effects, as opposed to other surface modeling techniques such as B-spline surfaces, and usually results in smooth shapes with evenly distributed vertices.

The most important parameter of the reaction constraints is the spring constant, which controls how rapidly the surface fulfills its constraint. This is usually set as high as possible without causing the simulation to become unstable, and should have a time constant significantly smaller than any other time constants in the model. It is also important to make sure that the skin surface does not fall through its constraint surface so that the muscle comes completely outside the skin. This can be avoided by assuring that muscle surface components are larger than the spacing between mass points on the skin surface and by not moving the skeleton too rapidly while the simulation is progressing. If the skin does fall through, it is necessary to turn off the reaction constraints, reinflate the skin, turn back on the reaction constraints, and again let out the air pressure.

The entire process of wrapping the skin around the muscles is therefore very much dependent on the history of the construction process performed while the simulation is in progress, and shows how much the elastic simulation itself is used as an interactive construction tool. Figure 4 illustrates how reaction constraints work to force the elastic surface, initially in the shape of a cylinder, to be outside a spherical constraint surface.

5.5. Binding The Skin Surface To The Muscle Layer

The effect of connective tissue, or fasciae, is simulated by creating attractive spring constraints between individual points on the skin and the muscle surface. To add these "rubber-band" force constraints [Witkin 87], the animator first places the skeleton in a neutral position while the simulation progresses so that the skin is well-distributed over the articulated figure. Then, some or all surface points can be bound to the underlying muscle layer, either manually on a point-by-point basis, or automatically. Manual attachment can be performed for each point individually by grabbing the skin surface point with the mouse and dragging it to the desired attach point on the muscle surface. Automatic attachment can be performed by selecting a group of points, using the mouse button, and giving the *attach* command which for each point traces a ray perpendicular to the skin surface to determine the attach point on the muscle layer.

The spring and damping constants of the rubber-bands can then be adjusted, either globally or for individual points, to give the desired tightness or looseness of skin attachment. Globally varying the spring constants affects the overall degree to which the skin clings to the muscle layer, altering the general appearance. By locally varying the spring constants, together with the local skin elasticity parameters, a variety of static and dynamic effects can be created such as skin folding, squash and stretch and follow-through. For example, in Figure 5 the skin above the navel was pulled in by increasing the spring constant of its connective tissue binding.

5.6. Sculpting the Fat Layer

The fat layer is modeled simply as a thickness below the skin layer. It is therefore implemented by offsetting the skin surface points by their fat thickness perpendicular to the skin surface and using these points as inputs to the reaction constraints. The thickness of the fat layer can be adjusted, either as a global parameter or by selecting individual points on the mesh using the mouse and setting their fat thickness locally. This allows the animator to control the shape of the surface to some extent simply by locally sculpting the fat layer. For example, in Figure 6, the fat thickness above the sacrum (highlighted in magenta) has been set to zero while the surrounding hip areas have a fat thickness of one centimeter, corresponding to the actual human anatomy. Globally altering the fat thickness is a simple way to make the character gain or lose weight.

5.7. Animation

Once the character has been constructed and all of its physical parameters defined, it may be animated simply by animating the skeleton. The motion of the skeleton provides the input forces which drive the skin surface. There exists a variety of dynamic and kinematic techniques for animating articulated figures. We have chosen a key-frame animation technique in which a series of key postures is specified and a smooth skeletal motion is interpolated between them [Girard 87]. The key postures are specified interactively using the inverse kinematics manipulator described in the previous section. Although the interpolated skeletal motion is a purely kinematic one, the resulting dynamic motion of the skin is physically simulated, resulting in a richer form of automatic inbetweening. For example, a perfectly cyclical skeletal motion such as a walk sequence will not necessarily result in a perfectly cyclical skin motion, but rather will vary somewhat from cycle to cycle, depending on the time constants of the elastic surface.

To animate the figure, the user positions the skeleton into a sequence of key postures, either without the elastic surface, or with the simulation running at a low surface resolution for interactive speed. A smooth motion can then be created by interpolating the joint angles using an interpolating spline [Kochanek 84]. The resulting skeleton motion may then be played back at any speed to check the animation, although if the simulation is running, this should not be too fast. To give an idea of what the final animation sequence will look like, the simulation can be turned off and then the skeleton motion sequence can be played back at full-speed. To get an accurate impression of the skin dynamics, the simulation can be turned back on while the skeleton motion is played back in simulation time. Key postures can be edited by selecting the particular key and repositioning the skeleton interactively.

5.8. Increasing the Surface Resolution

One of the advantages of using a rectangular mesh to represent the surface is that the mesh resolution can be changed quite easily. All the current values of the mesh (e.g. position, velocity, elasticity, spring constants) are bilinearly interpolated to determine the values of the higher resolution points. Although the character is usually designed and animated at a low surface resolution, once a motion sequence has been specified, the resolution can be increased and the same motion played back in simulation time to calculate a final motion sequence. This motion sequence is stored as a large array of successive elastic surface meshes and can be played back at interactive rates and viewed from different angles to check the final animation. Then the entire sequence can be rendered off-line using a standard rendering package.

Using unoptimized code on an SGI Indigo Extreme , we have been able to construct and animate full characters such as a penguin at a surface resolution of 16 x 16 mass points in one tenth real-time. With a redraw rate set to five frames per second, which is barely adequate for interactive work, about half the CPU time is spent redrawing the screen and half running the simulation. When scaled up to 32 x 32 mass points, the simulation slows down by a factor of eight to 1/80th real-time, which is still fast enough for calculating sequences for final rendering. For more complex character models, such as the torso in Figure 2, interactively manipulating the skeleton while the simulation is running becomes prohibitively slow, but interactive character construction is still practical.

6. LEMAN SYSTEM DESIGN

LEMAN was designed to allow rapid prototyping and experimenting with different types of user-interface style. In practice, when we add new functionality to the system, we usually first add a mouse-based interface to control it, using a slider widget or trackball metaphor, for instance. This can be done quite quickly using the interactive interface building tools of the Fifth Dimension Toolkit. Once this has been tested, we can then easily add other multi-dimensional input devices such as the spaceball, valuators, MIDI keyboard, or dataglove, simply by changing a run-time configuration file. This allows us to quickly change user-interface metaphors for experimentation.

6.1. Design Philosophy

Ideally, the entire system would have been written in an object-oriented language. However, this was not practical for several reasons, so the C language was chosen. The LEMAN system code therefore consists of three major components, each with its own style of programming. The first component implements the two-dimensional widgets and input devices and calls the Fifth Dimension Toolkit [Turner 90],which uses an object-oriented style of C. The second component is a set of purely numerical routines to implement all of the low level geometric and physically-based modeling calculations. These numerical routines were written in a traditional, FORTRAN-like style, with large numbers of parameters consisting of simple data structures such as arrays, and not allocating any dynamic memory.

The third system component is made up of a set of C modules organized in a manner similar to classes in an object-oriented language. These "classes" have constructor and destructor routines defined for them so that they can allocate memory to create "instances" which can be arranged in a typically object-oriented style of run-time data model. A limited form of attribute inheritance is permitted through the means of the macro preprocessor, although there is no message dispatching. This "object-oriented" portion of the code can be considered a higher level layer which is implemented on top of the 5D Toolkit and numerical routine libraries. The LEMAN classes themselves can be grouped into three categories, more or less along the lines of the standard MVC (Model-View-Controller) paradigm.

6.2. System Overview

From a run-time point of view, the LEMAN system can be viewed as a collection of distributed processes communicating through an interprocess communication protocol, as shown in Figure 7. For reasons of bandwidth limitation over the ethernet local area network, most of the numerical simulation and graphical rendering routines are contained in a single process which runs on a high-performance Silicon Graphics workstation. The other processes, which can be running on other machines, are used to implement much of the user interface portion of the software. This includes the 5D Toolkit widget panels, which provide the two-dimensional user interface component of the system. These panels are constructed interactively using FirstStep, the user interface building tool which comes as a standard application in the 5D Toolkit. Remote processes are also used to collect events from and send events to external input and output devices such as the MIDI keyboard, video recorder (using SMPTE time code) and audio synthesizer.

6.3. IPC Event Message Protocol

All of these processes communicate event information via simple byte streams according to an ascii (i.e. human readable) event protocol. Although this is rather inefficient, it allows easy monitoring of events as they pass between processes. It also allows certain events to be filtered out or altered according to a set of rules specified as a list of regular expression substitutions. These event filters, which are stored as files and loaded into the processes at run time, allow various input devices and widget panels to be bound to certain commands in the main application process without changing the source code. For example, a button with a certain name can be bound to the "quit" event or a particular key on the MIDI keyboard can be bound to a certain control parameter. Events can also be stored as command lines in a startup command file which is loaded in and executed at run time. Both event filter specification and event commands can be stored in the same file, called a command file (.cmd extension) which can be executed like a Unix shell script. The event protocol can therefore be thought of as a type of language that the LEMAN system uses to communicate events or control information between processes.

6.4. File Format

The system also has a standard ascii save file format. All LEMAN classes which represent some portion of the data model (the M part of the MVC structure) know how to store themselves to an ascii file along with pointers to their referenced instances, so a simple "save" routine called on the top-level object in the data model will recursively store the entire data model. The language is therefore a "flat" listing of each instance, with pointers providing the graph of instance relations. This is more general than a recursive type of language, which can normally only encode simple hierarchy and does not allow a general graph of relations to be represented. Within each object instance, attributes are encoded as keyword-value pairs, with the values having any number of parameters. Order of keywords is unimportant and unrecognized keywords are ignored. In this way a certain amount of forward and backward compatibility can usually be afforded between different versions of the file format.

6.5. Modeling Classes

The modeling classes represent the actual data model upon which the application program acts, similar to the document concept on the Macintosh. This, in practice, means that the modeling classes are distinguished by alone having load and save routines defined on them. In certain situations, modeling instances may be used to implement portions of the user interface, for example highlight materials or 3D icons, and these normally are not saved. Ideally, in keeping with the fine-grained MVC philosophy, the modeling classes should be completely passive repositories of data, having neither the ability to draw themselves nor the ability to respond to events. This would require separate viewers and controllers for each modeling class. For practical reasons, we have only made this separation of function in a few of the highest-level and most complicated modeling classes.

The diagram of the modeling hierarchy classes is shown in figure 9, using a style of object-relation diagram, described in figure 8. The heart of the modeling hierarchy is the node class. This object maintains local and global transformation matrices, front and back surface materials, and a texture, as well as viewing attribute information and joint angle information if it is not a fixed node. The node maintains a list of children, for implementing node hierarchy, and a pointer to a subnode, for maintaining subhierarchies. This is how links are represented, for example. The node can also contain a pointer to a model object. Like the material and texture objects, the models can be multiply referenced, but nodes normally cannot. The material objects maintain information about the node's surface material such as diffuse and specular reflectance. The texture object maintains a pointer to a texture bitmap file. The model object can be one of a variety of shapes, or a general rectangular or triangular mesh. The model class also maintains global deformation parameters.

The entire run-time data model for LEMAN system consists of the following components: For representing the skeleton, a strict DAG containing a hierarchy of node3D instances which may multiply reference lower level instances such as models, materials, and textures. Sitting next to the skeleton DAG, with pointers into it at many levels, are the multitrack instance, on one side, and the elastic surface instance on the other. Taken together these constitute a single animated layered elastic character, which may be saved and loaded at will. On top of the modeling data structure rest the controller and view instances, above which lies the single commander instance.

6.6. Maintaining Internal Consistency

Maintaining internal consistency of such a complicated data model is obviously a difficult task. Ideally, it would use some sort of constraint maintenance system such that mathematical or other relationships between objects could be declared, and then maintained using a predefined constraint maintenance algorithm. However, this was not available at the time. Therefore, the techniques used for maintaining the internal consistency of the LEMAN data model are by necessary ad hoc. They are not always as efficient as possible, but they are reasonably straightforward and simple.

The basic cycle of dynamic behavior in the system is the event loop. Events are removed from the queue and then handled. As each event is handled, data structures are updated as much as is determined necessary for subsequent events to be able to act. At periodic intervals, (e.g. five times per second) a clock tick event occurs which triggers a redraw routine. The draw operation first performs a more thorough updating operation on the entire data model so that it will be prepared for the subsequent rendering operation. When there are no events in the queue, an *idle_loop* operation is called continually as long as no event is found in the queue. This *idle_loop* operation performs a single iteration of the real-time motion control and physical simulation operations, which themselves require the data model to be updated to a certain degree, although not as thoroughly as for rendering.

The most important update operation is on the skeleton modeling DAG. This consists of a tree of node instances with multiple references to model, light, camera, material and texture instances. These latter classes maintain an updated flag which is only set to false when some internal parameter is changed by, for example, some interactive editing operation. It can be very difficult to tell when a node3D instance needs to be updated, however, so no updated flag is maintained. At times when it is necessary to be sure that the hierarchy is up-to-date, the entire node3D hierarchy (or subportions thereof) is updated in a recursive operation. This involves concatenating the path of local transformations to determine the global transformation. The inverse global transformations, which are normally used only in special operations such as inverse kinematics, are only updated on a "need-to-know" basis.

6.7. Event Handling

The 5D Toolkit can returns both local and remotely generated events in the form of an event protocol string containing the event information (consisting of type string, source name string, and data string). Since it is string based, this is not terribly efficient, but by programming all event handling using strings, the application program can be made to respond to both local and remote events transparently, and the full power of the IPC protocol and its event filtering are made available.

Event handling in LEMAN is therefore performed by examining these event protocol strings. At the top level, the *commander* object examines the source string of the event and distributes it to the various view and control instances according to the second level name. Event source strings take the form of hierarchical names, separated by dots, and by convention, the second level name of an event's source object is identified with its destination object's class name. In this way, events can be sent to the currently selected node3D instance, for example, by simply building a widget panel with the name "node3D". Any events coming from widgets in the panel will then be directed to the node controller object which will distribute them to its current node pointer. In this way, the user interface panels can be designed completely interactively and can be located either locally or on a remote machine.

Once the event has passed down the hierarchy to its target instance, it is handled by the object's *handle_event* routine. At this point, appropriate action can be taken by writing handler code for each event, or it can be handled automatically by the parameter handling facility which allows any numerical parameter attribute of a class to be declared at run-time along with an identifying name string. The parameter will then be updated automatically, according to the parameter and event types, when an event bearing its name is handled. This facility makes the job of putting a new control parameter under interactive control a simple two-step matter of declaring the parameter in the object's create routine and interactively placing an appropriate widget in the control panel.

6.8. Drawing

Before the rendering operation can be carried out, the entire graphical model must be updated completely. First the skeleton hierarchy is updated, starting from the root node. At each node, the current joint value is clamped to be within the bounds of its minimum and maximum values. Then the offset transform is determined from this value, depending on whether it is a rotational or translational node. The local transform is then premultiplied by the offset transform to determine the final transform. This is then postmultiplied by the global transform of the parent node to determine the global transform. The node then updates any modeling instances it may contain, which usually consists of updating the vertices and normals of its polygonal representation if any model parameters have changed. Next the elastic surface object is updated, which consists simply of recalculating its normal vectors, since any change to the elastic surface vertices is implemented by the evolve simulation routine, and is assumed up-to-date. The *draw* routine is then called on the node viewer and elastic surface viewer, which render the skeleton and skin surface respectively according to the current viewing parameters such as wireframe, highlighting selected nodes, etc.

6.9. Motion Control

Simple forward motion control of the skeleton can be performed by changing the angle of a single joint using a slider in the node controller window. The next update routine with therefore automatically do the forward kinematics calculations. Inverse kinematic motion control is implemented by the node controller object, using the delta_ik routine. This routine takes an end-effector node as an input parameter and constructs a kinematic chain from it to the node controller's current kinematic root node. It then allocates memory for and fills in the components of the Jacobian matrix. Each time the end effector is moved by a small amount, the Jacobian is recalculated and a differential joint motion is computed [Klein 83].

6.10. Attribute Inheritance

Node attribute inheritance is determined at draw-time, rather than update time. The reason for this is to allow multiple views of the same hierarchy with different viewing attributes. When the draw routine is called by the node view object, it passes along a pointer to an inheritance callback routine. The inherited attributes of each node are then determined by walking up the hierarchy until all undefined attributes have been resolved. At each level in the hierarchy, the inheritance callback routine is called which can override the inheritance for special nodes such as the currently selected node. This mechanism is used for highlighting special nodes and changing the colors of subhierarchies accordingly. It is also used interactively for selectively changing the visual attributes of portions of the hierarchy. This could probably be better implemented using a separate attributes object which would be used to determine the attribute inheritance at update time, if a suitable way could be found to allow multiple views.

6.11 Numerical Solution

The numerical algorithm used to solve the partial differential equation of motion for the elastic surface is a simple Gauss-Siedel or relaxation technique. In order to get maximum speed out of the system, only a few relaxation iterations are performed at each time step, typically two to four. This small number of iterations is usually sufficient because each successive solution is fairly close to the previous one, and because the resulting error in the solution manifests itself as damping in the physical system, which is usually desirable to control the oscillations of the skin surface.

Calculation of the reaction constraints is broken up into two stages. First, the entire elastic surface rectangular mesh is inside-tested against the skeleton model shapes to see if any of the points are within the muscle layer surfaces. If any of these points are found to be inside, a rectangular mesh of gradient vectors is returned, pointing towards the muscle surface. Then this gradient array is passed on to the reaction constraint routine. This routine takes the gradient array, together with the array of external forces, and calculates a new array of forces which enforce the reaction constraint.

The inside/outside test is performed by the inside_ routines which are defined on the node and model classes. When called on the root node of the character skeleton, this routine takes the array of elastic surface points and recursively inside-tests it with each of the models in the skeleton hierarchy. Within the model inside routine, the surface array is first transformed into local coordinates, then (if there are any deformations present) into the undeformed coordinate system.

At this time, the actual intersection test is performed for each point in the array against the surface shape. For spheres and cylinders, this is a simple radius test. For superellipses, this involves one invocation of the inside/outside function for each point. If this function is less than one, the point is inside the superellipse. The constraint gradient is then estimated by calculating the gradient value of the i/o function and multiplying by the difference of the i/o function from one. This gradient is then transformed back into local deformed and then global coordinates (making sure to use a covariant transformation) to yield the final constraint gradient vector array. This array, which essentially identifies each point which doesn't meet the constraint along with a direction in which to move to attempt to meet the constraint, is then ready to be passed on to the reaction constraint calculating routine.

6.12. Picking

The selection operation is usually done with the right mouse button and is implemented using the picking routines. Picking, that is finding out which object the cursor is over, is performed exclusively by ray-tracing. First the x and y position of the cursor in pixels is determined and from it a ray in screen coordinates is constructed. This ray is then converted to global coordinates based on the current viewing matrix and perspective projection. The ray is then passed down the skeleton node hierarchy, being transformed into local coordinates at each node, and intersection tested against each model primitive. The closest intersection point, i.e. with the smallest ray parametric coordinate, is returned along with a pointer to the node and any local model information about the point such as its u-v coordinates. The global position of the intersection point can be determined from the global ray. The skin rectangular mesh is also tested for intersection by exhaustively testing each polygon in the mesh. The same ray-tracing routines are also used by the automatic binding algorithm for finding near-by attach points on the muscle surface.

7. CONCLUSION

The LEMAN system was designed not only to test new 3D character models, but also to see if the types of direct manipulation interaction metaphor used in 2D could be generalized to a complex 3D domain such as character animation. Despite the greater sophistication of three-dimensional application domains, and the resulting complications involved in designing as well as using such software systems, the results of these experiments are definitely positive. Almost every type of 2D direct-manipulation technique can work as easily in the three dimensional domain. By using the two-handed "ball-and-mouse" metaphor, the user can use the left hand (for example) to position the character in any desired manner for selection or manipulation by the right hand. This effectively extends the dimensionality of the mouse as an input device, since it can select any exterior point on the character or move in two-dimensions within any arbitrary plane selected with the spaceball. Since much more information can be packed into a three-dimensional volume than onto a two dimensional plane, we can see that direct manipulation metaphors in three dimensions have the potential to make more efficient use of available screen space and increase the bandwidth of the human computer interface.

Direct manipulation of a 3D object can be contrasted, in terms of information content, to using widget-based interaction with many overlapping windows. In both cases, the information is stored in three dimensions and the screen simply shows a two-dimensional slice through the data. In the overlapping windows case, one must search the three-dimensional space by laboriously popping up different windows until the desired one is found. In the 3D direct-manipulation case, we are able to view a perspective rendering of the three-dimensional information space and move about freely within it, observing it from any angle. Since this is closer to what we do in the real world, we are better equipped to interact with a computer in this way.

A second, and somewhat surprising conclusion arrived at from building and using the LEMAN system is the interactive modeling capabilities presented by direct manipulation of a physical simulation in real time. The original attraction to physically-based techniques was because of its potential to enhance the quality of animation by making it more natural. It was therefore surprising to find out that it also made interaction more natural, and often made what would have been complex interactive tasks quite simple. For example, one important interactive task in building a layered elastic character is binding the skin to the muscle layer. This requires finding a mapping from every point on the elastic surface to a point on the muscle layer, a layer which has no topological similarity at all with a rectangular mesh surface. Using the LEMAN system, however, this turned out to be quite a simple task, as long as the physical simulation is running. By wrapping the elastic surface around the muscle layer initially and shrinking it with the reaction constraints turned on, the skin surface eventually finds an energy minimum where it is more-or-less equally distributed around the muscle layer. Binding therefore simply consists of attaching each skin mass point to its nearest perpendicular muscle surface point. By adjusting the position of the articulated figure when the binding operation is done, different types of skin attachments can be made. This kind of physically-based interaction technique may have considerable potential not only in animation, but in other fields as well, for example in general surface modeling.

These experiences reinforced the belief that layered elastic models, such as the elastic surface layer model, are a promising approach to constructing animated three-dimensional characters. By using 3D interactive techniques to manipulate the physical model as the simulation progresses, an animator can rapidly build and animate complex characters. Making the computer a genuinely useful and creative tool in the hands of a character animator requires a variety of modeling techniques combined in the right way and manipulated using the proper interaction metaphors. How to do this can only be determined by experimenting with the various possibilities. By building test systems such as LEMAN, in which various types of interactive construction and animation techniques can be explored, practical software tools for creating expressive character animation can be built.

The LEMAN system is a prototype, however, and does not yet present a practical system for 3D character animation. One of the main limitations, in particular, is the finite difference mesh, which has topological restrictions making it difficult to create surfaces with thin appendages (like arms and legs). Moving to a finite element discretization should remove these restrictions. Addition of self-collision detection to the skin would allow greater deformations at the joints and more pronounced wrinkling. Adding dynamic properties to other layers such as the fat and muscle layers would also enhance realism, as well as using some more advanced skeleton animation methods.

ACKNOWLEDGMENTS

The author is indebted to Enrico Gobbetti, Francis Balaguer and Daniel Thalmann for ideas, suggestions and 5D Toolkit software tools. The author also would like to thank Prem Kalra, Ying Yang, Tsuneya Kurihara, Geoff Wyvill, and Tat-Seng Chua for valuable discussions, Serge Rezzonico for FirstStep, Jim Basam for the Wavefront file format export, and Michael Eringis for Wavefront rendering. This work was supported in part by a grant from the Swiss National Research Foundation.

REFERENCES

1. Barr AH (1981) Superquadrics and Angle Preserving Transformations. *IEEE Computer Graphics and Applications,* Vol. 1, No 1:11-23

2. Barr AH (1984) Global and Local Deformations of Solid Primitives. *Proc.SIGGRAPH '84, Computer Graphics*, Vol. 18, No3:21-30

3. Carignan M, Yang Y, Magnenat-Thalmann N,Thalmann D (1992) Dressing Animated Synthetic Actors With Complex Deformable Clothes. *Proc.SIGGRAPH'92, Computer Graphics,* Vol. 26, No2: 99-104

4. Chadwick J, Haumann DR, Parent RE (1989) Layered Construction for Deformable Animated Characters, Proc. SIGGRAPH '89, Computer Graphics, Vol. 23, No3, pp.234-243

5. Chadwick J, Haumann DR, Parent RE(1989) Layered Construction for Deformable Animated Characters. *Proc. SIGGRAPH '89*, *Computer Graphics*, Vol. 23, No3 :234-243

6. Chen DT, Zeltzer D (1992) Pump It Up: Computer Animation of a Biomechanically Based Model of Muscle Using the Finite Element Method, Proc. SIGGRAPH'92, Computer Graphics, Vol. 26, No 2, pp.89-98

7. Culhane S (1988) Animation From Script To Screen, St. Martin's Press, New York 1988

8. Feynman RP, Leighton RB, Sands M (1965) The Feynman Lectures on Physics, Addison-Wesley, Reading Massachusetts 1965

9. Gascuel MP, Verroust A, Puech C (1991) A Modelling System for Complex Deformable Bodies Suited to Animation and Collision Processing, The Journal of Visualization and Computer Animation, Vol. 2, No3, pp.82-91

10. Girard M(1987) Interactive Design of 3D Computer-Animated Legged Animal Motion. *IEEE Computer Graphics and Applications,* Vol. 7, No6:39-51

11. Gourret JP, Magnenat-Thalmann N, Thalmann D (1989) Simulation of Object and Human Skin Deformations in a Grasping Task, Proc. SIGGRAPH '89, Computer Graphics, Vol. 23, No 3, pp.21-30

12. Klein H, Review of Pseudoinverse Constrol for Use with Kinematically Redundant Manipulators, IEEE Transaction on Systems, Man and Cybernetics Vol SMC-132, No. 3, March/April '83

13. Kochanek DH, Bartels RH (1984) Interpolating Splines with Local Tension, Continuity, and Bias Control, Proc. SIGGRAPH '84, Computer Graphics, Vol. 18, pp.33-41

14. Magnenat-Thalmann N, Laperrière R, Thalmann D (1988) Joint-Dependent Local Deformations for Hand Animation and Object Grasping, Proc. Graphics Interface '88, pp.26-33.

15. Miller G(1988) The Motion Dynamics of Snakes and Worms. *Proc. SIGGRAPH '88, Computer Graphics,* Vol. 22, No4:169-173

16. Overveld CWAM (1990) A Technique for Motion Specification in Computer Animation, The Visual Computer, Vol. 6, No 2, pp.106-116

17. Platt JC, Barr AH (1988) Constraint Method for Flexible Models, Proc.SIGGRAPH '88, Computer Graphics, Vol. 22, No4, pp.279-288

18. Sederberg TW, Parry SR (1986) Free-Form Deformations of Solid Geometric Models, Proc. SIGGRAPH'86 Computer Graphics, Vol. 20, No4, pp.151-160

19. Terzopoulos D, Platt JC, Barr AH, Fleischer K (1987) Elastically Deformable Models, Proc.SIGGRAPH'87, Computer Graphics, Vol. 21 No 4, pp.205-214

20. Terzopoulos D, Waters K (1991) Techniques for Realistic Facial Modeling and Animation, in: Magnenat Thalmann N, Thalmann D, Computer Animation '91, Springer-verlag, Tokyo, pp.59-74

21. Tu X, Terzopoulos D (1994) Artificial Fishes: Physics, Locomotion, Perception, Behavior, Proc. SIGGRAPH '94, Computer Graphics Annual Conference Series, ACM, pp.43-50

22. Turner R, Gobbetti E, Balaguer F, Mangili A, Thalmann D, Magnenat-Thalmann N (1990) An Object-Oriented Methodology Using Dynamic Variables for Animation and Scientific Visualization, in: Chua TS, Kunii TL, CG International '90, Springer, Tokyo

23. Turner R, Thalmann D. (1993) The Elastic Surface Layer Model for Animated Character Construction *Proceedings Computer Graphics International 1993,* Springer-Verlag

24. Walters G. (1989) The Story of Waldo C. Graphic, 3D Character Animation By Computer, SIGGRAPH '89 Tutorial Notes.

25. Waters K(1990) Modeling 3D Facial Expression Dynamic Muscle Models Control for Facial Animation. *SIGGRAPH'90 Tutorial*: 26 State of the Art in Facial Animation, Dallas, Texas

26. Witkin A, Fleischer K, Barr AH (1987) Energy Constraints on Parameterized Models, Proc. SIGGRAPH'87, Computer Graphics, Vol. 21, No4, pp.225-232

Russell Turner is Assistant Professor in the Computer Science Department at the University of Maryland Baltimore County. He received his Ph.D. in Computer Science in 1993 from the Swiss Federal Institute of Technology, Lausanne. His research interests include 3D interaction, character animation, object-oriented graphics and physically-based modeling.

Russell Turner
Computer Science Department
University of Maryland, Baltimore County
5401 Wilkens Avenue
Baltimore, MD 5401 Wilkens Avenue

phone: 01/410-455-3965
fax: 01/410-455-3969
http://www.cs.umbc.edu/~turner
email: turner@cs.umbc.edu

Figure 1. Stages in the Character Construction Process.

Figure 2. Venus Figure. Wavefront Rendering by Michael Eringis.

Figure 3. Building the Skeleton and Muscle Layer.

Figure 4. Reaction Constraints.

Figure 5. Binding Skin to Muscle Layer.

Figure 6. Sculpting Fat Layer.

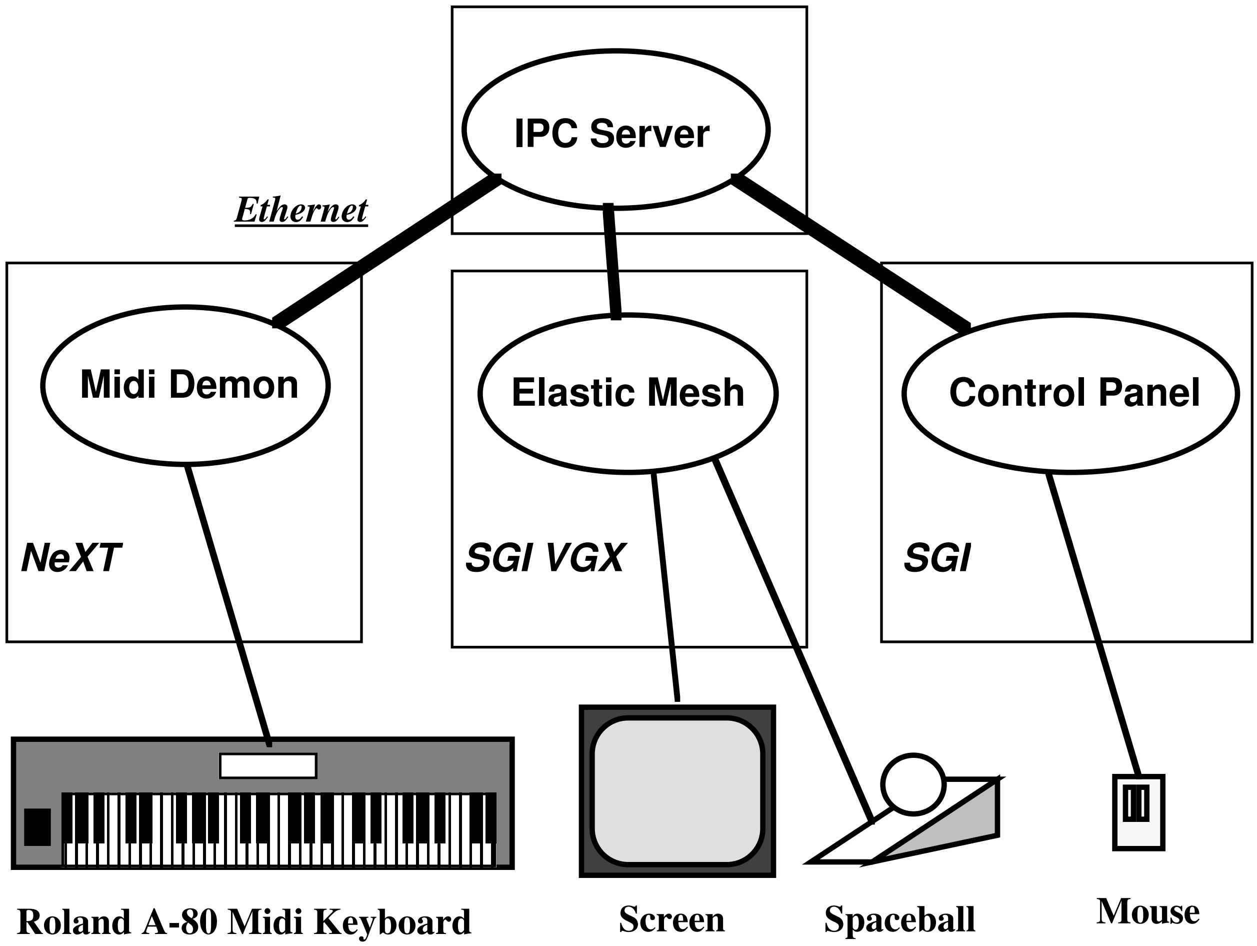

Figure 7: Cooperating Processes

A ——▶ B		Class A inherits from class B
A ——o B		Every instance of A is related to zero to one instances of B
A —— B		Every instance of A is related to exactly one instance of B
A ——● B		Every instance of A is related to zero to many instances of B

Figure 8: Object-Relation Diagrams

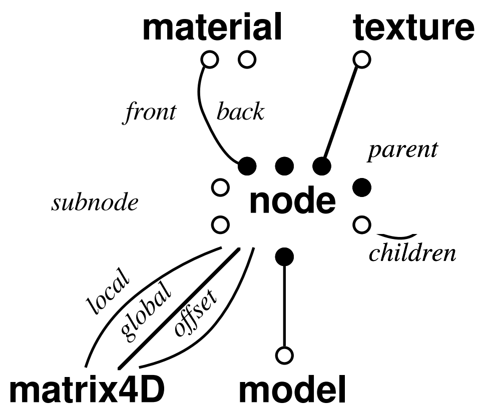

Figure 9: Modeling Classes